Tecnológico de Monterrey

Actividad 1 (Velocidades Lineales y angulares)

Bruno Manuel Zamora García A01798275

19 de febrero del 2024

Materia:

Fundamentación de robótica (Gpo 101)

Profesor

Alfredo García Suárez

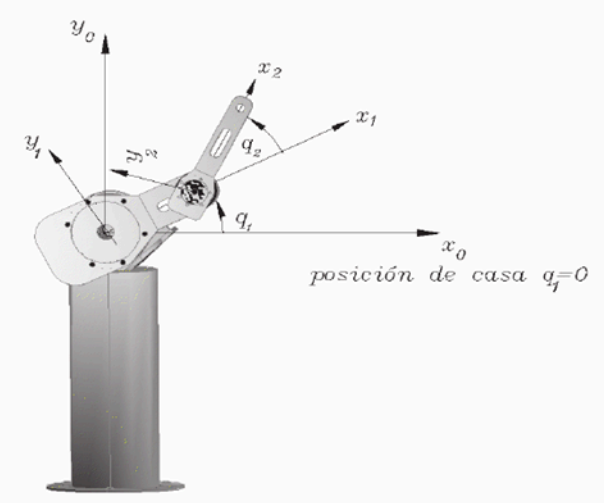
Introducción:

Este reporte describe el procedimiento que se llevó a cabo en MATLAB; paso a paso para obtener la velocidad lineal y angular de un robot planar manipulador de dos grados de libertad con articulaciones rotacionales. Se usa tanto el método diferencial como el método analítico para el cálculo del Jacobiano. El robot consta de dos articulaciones rotacionales, y se asume que las longitudes de los eslabones son conocidas.Finalmente, se usa el Jacobiano para determinar las velocidades del efector final.

Descripción del Robot

El robot analizado es un manipulador planar con dos articulaciones rotacionales:

- **Grado de libertad 1 (q1):** Rotación en la base.
- **Grado de libertad 2 (q2):** Rotación del segundo eslabón respecto al primero.
- **Longitudes de los eslabones:** l_1 y l_2
- **Sistema de referencia:** Se establece un sistema de coordenadas en la base del robot, con el eje x_0 alineado con el primer eslabón en su posición inicial.



Descripción para las obtención de velocidades del efector final

##Nota: lo que está en verde, es parte del código de matlab

1. Limpieza del entorno de trabajo

Se limpia el entorno de trabajo para evitar conflictos con variables o figuras anteriores.

```
% Limpieza de pantalla
clear all
close all
clc
```

2. Declaración de variables simbólicas

Se declaran las variables simbólicas necesarias para representar articulaciones y las longitudes de los eslabones.

```
% Declaración de variables simbólicas
syms th1(t) th2(t) l1 l2 t
```

-th1(t) y th2(t): Ángulos de las articulaciones rotacionales en función del tiempo.

-l1 y l2: Longitudes de los eslabones 1 y 2, respectivamente.

-t: Variable simbólica que representa el tiempo.

3. Configuración del robot

Definición del tipo de articulaciones del robot y la cantidad de articulaciones.

```
% Configuración del robot, 0 para junta rotacional, 1 para junta prismática
RP = [0, 0]; % Ambas articulaciones son rotacionales
```

-RP: Vector que indica el tipo de articulación. 0 para rotacional y 1 para prismática.

4. Vector de coordenada articulares

Se crea el vector de coordenadas articulares Q, este tendrá los ángulos de cada articulación.

```
% Creamos el vector de coordenadas articulares
```

```
Q = [th1(t); th2(t)];
disp('Coordenadas articulares');
pretty(Q);
```

-Q: Vector columna con las coordenadas articulares th1(t) y th2(t).

5. Vector de velocidades articulares

Se calcula el vector de velocidades articulares Qp

```
% Creamos el vector de velocidades articulares
Qp = diff(Q, t); % Utilizo diff para derivadas cuya variable de referencia no depende de otra: ejemplo el tiempo
disp('Velocidades articulares');
pretty(Qp);
```

-Qp: Vector columna que contiene las derivadas de th1(t) y th2(t) respecto al tiempo.

-diff(Q, t): Calcula la derivada de Q respecto a t.

6. Número de grados de libertad

Determinacion del numero de grados de libertad (GDL)

```
% Número de grado de libertad del robot
GDL = size(RP, 2); % Siempre se coloca 2, ya que indica la dimensión de las columnas
GDL_str = num2str(GDL); % Convertimos el valor numérico a una cadena de carácteres tipo string
```

-GDL: Número de grados de libertad, que en este caso es 2.

7. Cinemática directa

En esta sección se calculan las matrices de transformación homogénea locales y globales para cada articulación.

7.1 Articulación 1

-Posición de la junta 1 respecto al origen y Matriz de rotación de la articulación 1 respecto al origen:

```
P(:,:,1) = [l1 * cos(th1); l1 * sin(th1); 0];
R(:,:,1) = [cos(th1) -sin(th1) 0; sin(th1) cos(th1) 0; 0 0 1];
```

- *Función: Calcula la posición (P) y la matriz de rotación (R) de la primera articulación respecto al sistema de referencia base.*

- *Razón: Estas matrices describen la posición y orientación del primer eslabón, necesarias para construir las matrices de transformación homogénea.*

7.2 Articulación 2

-Posición de la junta 2 respecto a la 1 y Matriz de rotación de la articulación 2 respecto a la 1:

```
P(:,:,2) = [l2 * cos(th1 + th2); l2 * sin(th1 + th2); 0];
R(:,:,2) = [cos(th1 + th2) -sin(th1 + th2) 0; sin(th1 + th2) cos(th1 + th2) 0; 0 0 1];
```

- Función:Calcula la posición y la matriz de rotación de la segunda articulación respecto a la primera.

- Razón:Estas matrices describen la posición y orientación del segundo eslabón, necesarias para construir las matrices de transformación homogénea.

7.3 Matrices de transformación homogénea

-Matriz de transformación homogénea local y global:

```
% Creamos un vector de ceros
Vector_Zeros = zeros(1, 3);

% Inicializamos las matrices de transformación Homogénea locales
A(:,:,1) = simplify([R(:,:,1) P(:,:,1); Vector_Zeros 1]);
A(:,:,2) = simplify([R(:,:,2) P(:,:,2); Vector_Zeros 1]);

% Inicializamos las matrices de transformación Homogénea globales
T(:,:,1) = A(:,:,1);
T(:,:,2) = simplify(T(:,:,1) * A(:,:,2));
```

- Función:

A: Matrices de transformación homogénea locales.
T: Matrices de transformación homogénea globales.

- Razón:Las matrices de transformación homogénea combinan rotación y traslación en una sola matriz. Las globales describen la posición y orientación de cada articulación respecto al sistema base.

7.4 Vectores de posición y matrices de rotación globales

-Vectores de posición globales y matrices de rotación globales:

```
PO(:,:,1) = P(:,:,1);
PO(:,:,2) = T(1:3,4,2);
```

```
RO(:,:,1) = R(:,:,1);
RO(:,:,2) = T(1:3,1:3,2);
```

- Función:

PO: Vectores de posición globales.

RO: Matrices de rotación globales.

- Razón:Estas matrices y vectores describen la posición y orientación de cada articulación respecto al sistema de referencia inercial.

7.5 Matrices de transformación locales y globales

Se muestran las matrices de transformación:

```
for i = 1:GDL
    i_str = num2str(i);
    % Locales
    disp(strcat('Matriz de Transformación local A', i_str));
    pretty(A(:,:,i));

    % Globales
    disp(strcat('Matriz de Transformación global T', i_str));
    pretty(T(:,:,i));

    % Obtenemos la matriz de rotación "RO" y el vector de translación PO de la
    % matriz de transformación Homogénea global T(:,:,GDL)
    RO(:,:,i) = T(1:3,1:3,i);
    PO(:,:,i) = T(1:3,4,i);
    pretty(RO(:,:,i));
    pretty(PO(:,:,i));
end
```

8. Cinemática diferencial

Se calcula el jacobiano lineal y angular utilizando el metoda diferencial y analitico

8.1 Jacobiano lineal (método diferencial) o por derivadas parciales

```
% Calculamos el jacobiano lineal de forma diferencial
disp('Jacobiano lineal obtenido de forma diferencial');
% Derivadas parciales de x respecto a th1 y th2
Jv11 = functionalDerivative(PO(1,1,GDL), th1);
Jv12 = functionalDerivative(PO(1,1,GDL), th2);
% Derivadas parciales de y respecto a th1 y th2
```

```
Jv21 = functionalDerivative(PO(2,1,GDL), th1);
Jv22 = functionalDerivative(PO(2,1,GDL), th2);
% Derivadas parciales de z respecto a th1 y th2
Jv31 = functionalDerivative(PO(3,1,GDL), th1);
Jv32 = functionalDerivative(PO(3,1,GDL), th2);

% Creamos la matríz del Jacobiano lineal
jv_d = simplify([Jv11 Jv12;
                 Jv21 Jv22;
                 Jv31 Jv32]);
pretty(jv_d);
jv_d = simplify([Jv11 Jv12; Jv21 Jv22; Jv31 Jv32]);
```

- Función:Calcula el Jacobiano lineal derivando la posición del efector final respecto a las coordenadas articulares.

- Razón:El Jacobiano lineal relaciona las velocidades articulares con la velocidad lineal del efector final.

8.2 Jacobiano lineal y angular(metodo analitico)

Este segmento de código calcula el Jacobiano lineal (Jv_a) y el Jacobiano angular (Jw_a) utilizando un enfoque analítico, que es más eficiente que el método diferencial porque evita el cálculo de derivadas simbólicas y se basa en la geometría del robot.

```
% Calculamos el jacobiano lineal y angular de forma analítica
% Inicializamos jacobianos analíticos (lineal y angular)
Jv_a = sym(zeros(3, GDL));
Jw_a = sym(zeros(3, GDL));

for k = 1:GDL
    if (RP(k) == 0) % Casos: articulación rotacional
        % Para las articulaciones rotacionales
        try
            Jv_a(:,k) = cross(RO(:,3,k-1), PO(:,:,GDL) - PO(:,:,k-1));
            Jw_a(:,k) = RO(:,3,k-1);
        catch
            Jv_a(:,k) = cross([0,0,1], PO(:,:,GDL)); % Matriz de rotación de 0 con respecto a
0 es la Matriz Identidad, la posición previa también será 0
            Jw_a(:,k) = [0,0,1]; % Si no hay matriz de rotación previa se obtiene la Matriz
identidad
        end
    elseif (RP(k) == 1) % Casos: articulación prismática
        % Para las articulaciones prismáticas
```

```
        try
            Jv_a(:,k) = RO(:,3,k-1);
        catch
                Jv_a(:,k) = [0,0,1]; % Si no hay matriz de rotación previa se obtiene la Matriz
identidad
        end
        Jw_a(:,k) = [0,0,0];
    end
end
```

El código funciona iterando sobre cada articulación del robot (desde la primera hasta la última, según el número de grados de libertad. Para cada articulación, se verifica si es rotacional o prismática, ya que el cálculo del Jacobiano cambia según el tipo de articulación.

Para articulaciones rotacionales:

- El Jacobiano lineal se calcula utilizando el producto vectorial entre el eje de rotación de la articulación (z_{k-1}) y el vector que va desde la articulación actual hasta el efector final ($p_{efector} - p_{k-1}$). Este producto vectorial representa cómo la rotación de la articulación afecta la velocidad lineal del efector final.

- El Jacobiano angular es simplemente el eje de rotación de la articulación (z_{k-1}), ya que la velocidad angular del efector final está directamente alineada con este eje.

Para articulaciones prismáticas:

- El Jacobiano lineal es directamente el eje de movimiento de la articulación (z_{k-1}), ya que en una articulación prismática el movimiento es lineal y no rotacional.

- El Jacobiano angular es un vector nulo (0), porque las articulaciones prismáticas no generan rotación y, por lo tanto, no contribuyen a la velocidad angular del efector final.

Este enfoque analítico es eficiente porque utiliza la geometría del robot (posición y orientación de articulaciones) para calcular el Jacobiano, en lugar de derivar expresiones simbólicas.

8.3 Se muestran los Jacobianos

```
Jv_a = simplify(Jv_a);
```

```
Jw_a = simplify(Jw_a);
disp('Jacobiano lineal obtenido de forma analítica');
pretty(Jv_a);
disp('Jacobiano ángular obtenido de forma analítica');
pretty(Jw_a);
```

9. Calculo de velocidad lineal y angular

Se calculan las velocidades lineal y angular usando los Jacobianos anteriores

-Velocidad lineal: $V = J_V * Q$

-Velocidad angular: $W = J_W * Q$

```
disp(size(Q))
disp(size(Qp))

disp('Velocidad lineal obtenida mediante el Jacobiano lineal');
V = simplify(Jv_a * Qp);
pretty(V);
disp('Velocidad angular obtenida mediante el Jacobiano angular');
W = simplify(Jw_a * Qp);
pretty(W);
```

Para obtener las velocidades lineal (V) y angular (W) del efector final, se multiplican los Jacobianos lineal (Jv) y angular (Jw) por el vector de velocidades articulares (). Esto se debe a que los Jacobianos representan cómo las velocidades de las articulaciones afectan el movimiento del efector final.

- Jacobiano lineal (Jv): Relaciona las velocidades articulares con la velocidad lineal del efector final.

- Jacobiano angular (Jw): Relaciona las velocidades articulares con la velocidad angular del efector final.

Esta multiplicación funciona porque los Jacobianos usan la contribución de cada articulación al movimiento del efector final, y al multiplicarlos por las velocidades articulares, se obtiene el efecto combinado de todas las articulaciones en el espacio cartesiano.

10.Conclusión

En este reporte, plantee el proceso paso a paso para calcular las velocidades lineal y angular del efector final de un robot planar de dos grados de libertad con articulaciones rotacionales. Para realizar el código se usaron los métodos de cinemática directa para determinar la posición y orientación del efector final, y cinemática diferencial para calcular los Jacobianos .Dichos Jacobianos permitieron relacionar las velocidades articulares con las velocidades del efector final.

MATLAB fue fundamental para realizar cálculos simbólicos y numéricos de manera más rápida y precisa, MATLAB me permitió visualizar las matrices de transformación homogénea, los Jacobianos y las velocidades resultantes. El enfoque analítico demostró ser más eficiente que el diferencial, ya que evita el cálculo de derivadas simbólicas y se basa en la geometría del robot.